JAVA

10 Days in Java ...

بسم الله الرحمن الرحيم

~(وَقُلِ اعْمَلُوا فَسَيَرَى اللَّهُ عَمَلَكُمْ وَرَسُولُهُ وَالْمُؤْمِنُونَ)~

في مثل هذه اللحظات يتوقف اليراع ليفكر قبل أن يخط الحروف ليجمعها في كلمات ... تتبعثر الأحرف وعبثاً أن يحاول تجميعها في سطور ... سطوراً كثيرة تمر في الخيال ولا يبقى لنا في نهاية المطاف إلا قليلاً من الذكريات وصور تجمعنا برفاق كانوا إلى جانبنا فواجب علينا شكرهم ووداعهم ونحن نخطو خطوتنا الأولى في غمار الحياة ...

وأقوم بجزيل الشكر والعرفان إلى كل من أشعل شمعة في دروب عملنا وإلى من وقف على المنابر وأعطى من حصيلة فكره لينير دربنا ... إلى الأساتذة الكرام في كلية الملك عبد الله الثاني لتكنولوجيا المعلومات واخص بذلك، من علمني لغة الجافا :

م. أصيل العناني   م . رولا الخالد

هذه كلماتي قد عبرت عن بعض أحاسيسي وما قدمتموه لنا من عطاء ...

وأعذر قلم جف حبره باسم المودة والاحترام وباسم كل من وافق على حروفي عن التقصير ...

Ehab A. Qad'oumi ...

Content :

❖ Arrays

- Declaring and Creating Arrays
- Multidimensional Arrays
- Sorting Array.

❖ Object Oriented Programming: Inheritance .

- Super classes and Subclasses
- Relationship between Super classes and Subclasses
- Inheritance

❖ Strings

- Declaration Strings
- Strings methods

❖ Exception Handling .

- Exception-Handling Overview
- Types of exceptions
- Java Exception Hierarchy
- finally block

❖ Java Applet

- Introduction in java applet
- Applet methods
- Color Class
- Examples in applet

Introduction to Java Programming Language

☒ What is programming ?

Programming means : a set of commands and instructions given to a computer in the form of a program written in a particular programming language by a word processor program source consists of several lines and each line is considered among computer and deal with each sentence in a particular order to accomplish the program which is designed to achieve , **Java** is one of the programming languages you create an application .

☒ What programming requirements ?

- ***Interpreter Interpreter :***

Some programming languages require an interpreter to interprets each line of the program and tell the computer tasks to be do, and this language by language and characterized such as Basic languages that need explaining easily tested but flawed because they are slow when run .

- **Compiler Compiler :**

Requires others of programming languages interpreter translates the program and converted into a form understood by the computer and Programs are translated quickly run but flawed need more time to test where he writes software then translates and then try In case of errors must be corrected first and then re-translated and then tested to verify the demise error and unique **Java language requires a translator and interpreter .**

☒ History of java :

Java **fabricated** with James Gosling, Patrick Naughton, Chris Warth, Ed Frank, And Mike Sherid . They founded the **sun microsystem** and had its beginnings in 1991 fired on the language in a timely manner **Oak** name and after the development took **18 months** to launch its first Version in 1992 and renamed in **1995** to **Java** .

Over time several version of Java were released which enhanced the language and its libraries. The current version of Java is Java 1.6 also known as Java 6.0.

☒ Overview :

Java programming language consists of a **Java compiler**, the **Java virtual machine**, and the **Java class libraries**. The Java virtual machine (JVM) is a software implementation of a computer that executes programs like a real machine. The Java compiler translates Java coding into so-called byte-code.

The Java virtual machine interprets this byte-code and runs the program. The Java virtual machine is written specifically for a specific operating system. The Java runtime environment (JRE) consists of the JVM and the Java class libraries.

☒ Types Of programming Languages :

1. Machine language.
2. Assembly language .
3. **High level language.**

☒ Characteristics of Java :

The target of Java is to write a program once and then run this program on multiple operating systems.

- **Platform independent:** Java programs use the Java virtual machine as abstraction and do not access the operating system directly. This makes Java programs highly portable. A Java program which is standard complaint and follows certain rules can run unmodified all several platforms, e.g**. Windows or Linux.**
- **Object-orientated programming language**: Except the primitive data types, all elements in Java are objects.
- **Secure:** java code convert to byte code during compilation , and it does not work with low level programming (assembly language) which is the common language for writing viruses **.**
- **Portable:** Java programs run on more than one operating system at the same time .
- **Error handling :**Possibility to resume execution even if grammatical error occurred (syntax error) in the middle of the program.
- **Interpreted and compiled language:** Java source code is transferred into byte-code which does not depend on the target platform. This byte-code will be interpreted by the Java Virtual machine (JVM). The JVM contains a so called Hotspot-Compiler which translates critical byte-code into native code.

*The Java syntax is similar to **C++**. Java is case sensitive, e.g. the variables my Value and my value will be treated as different variables.*

☒ Development with Java:

The programmer writes Java source code in a text editor which supports plain text. Normally, We use Textpad ,Eclipse , Netbeans editors …

You do not need any other development tool, such as an Integrated Development Environment (**IDE**). We strongly recommend that you not use anything but a basic text editor until you complete this book . An IDE can protect you from some of the details that really matter, so you're much better off learning from the command-line or textpad Editor and then, once you really understand what's happening, move to a tool that automates some of the process …

At some point the programmer calls the Java compiler (javac). The Java compiler creates platform independent code which is called byte-code. T**his byte-code is stored in ".class" files.**

Byte-code can be executed by the Java runtime environment. The Java runtime environment (JRE) is a program which knows how to run the byte-code on the operating system. The JRE translates the byte-code into native code and executes it, e.g. the native code for Linux is different then the native code for Windows.

By default, the compiler puts each class file in the same directory as its source file.

☒ Types of java:

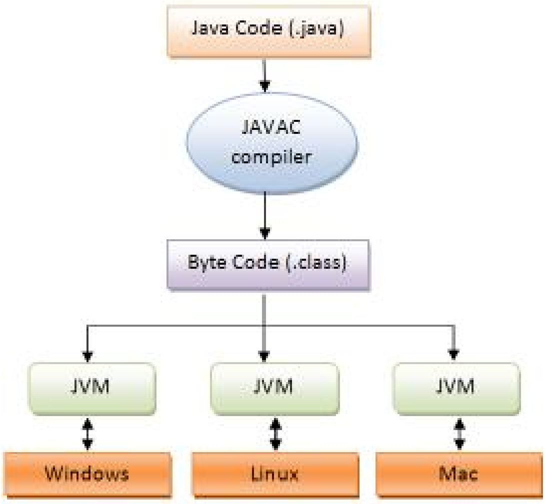
- Java Standard Edition (Java **SE**) : use to designs Desktop application.
- Java Enterprise Edition (Java **EE**) : applications and web applicationsJVM,
- Java Micro Edition (Java **ME**) : geared toward applications for small, memory constrained devices

Introduction to Java Applications

Note that : In this Course , I will Discussing **Standard Edition** of java !

⊠ Phases for execute java program :

1) Write a java source code and store it by "**NAME.java**".
2) Simple Run java program: the compiler create the byte code and store them in a file with extension ".class".
3) Loading the program in the memory.
4) Byte code verification.
5) Execution by JVM.



⊠ Program style in java :

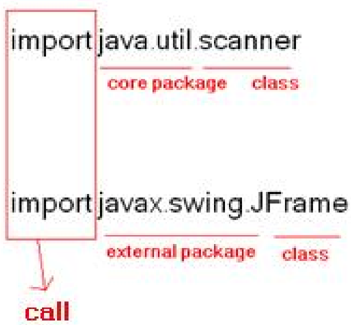
- **application** : program that execute from the user "Local Computer"
  ➔ there are main method , the basic unit to write a java program is **Class**
- **applet** : for designing graphics …
  ➔ there are no main method

⊠ Two types of classes :

- API (application programming Interfaces) :group package
  ➔ collection of pre-defined class
- User defined class.
  ➔ a classes that user can be declaration such as a variables and methods .

⊠ packages two types :

- **Core** : java.util ,…
- **external** : javax.swing , …



Ehab A. Qad'oumi …

☒ General Structure of the Java application:

→ The General Syntax in java application

```
public class CLASSNAME
{
        public static void main (String [] arg)
        {

        } // end main method

} // end of public class
```

Notes:

→ The name of the program must be the same name as the public class

→The program must contain a public class and only one

→The JVM to start the implementation of the program of the main method and anything outside the main method will be implemented only if summoned

---

☒ Displaying text with print , println and printf

- System.out.print(……);
- System.out.println(……);
- System.out.printf(……);

- System.out.**print**(……);

print in line , when you end printing the line , the curser still in the same line

Ex:
System.out.print("Welcome to my first program ! ");
output→ Welcome to my first program !

Ex:
System.out.print("10"+10);
output→ 1010

- System.out.**println**(……);

print in line but , when you end printing the line , the curser go to the next line.

Ex:
System.out.println("Welcome to my first program ! ");
output→ Welcome to my first program !

But, the cursor go to next line !!!

Ex:
System.out.println("Hello ");
System.out.println("you aare in java lesson");
output→ Hello
you aare in java lesson

- System.out.**printf**("**FORMAT STRING**","String to print");

this type put an format to print anything you want …

- %s →Strings .
- %S →Strings , but convert all letters to capital letters .
- %d →integers
- %f → float
- %B → Boolean

Ex:
System.out.printf ("%S ","Welcome to my first program! ");
output→ WELCOME TO MY FIRST PROGRAM !

Ex:
System.out.printf ("%f ", 2.0);
output→2.000000

every **specifire** has only one block to print it.

Ex:
System.out.printf ("%d add %d= %d ",10 , 20 , (10+20));
output→10 add 20 = 30

Ex:
System.out.printf ("%s " , "java" ,"application");
output→ java

☒ Special escape:

Java language supports few **special escape** sequences for String and char literals as well. They are:

Notation	Character represented
\n	Newline (0x0a)
\r	Carriage return (0x0d)
\f	Formfeed (0x0c)
\b	Backspace (0x08)
\s	Space (0x20)
\t	tab
\"	Double quote
\'	Single quote
\\	backslash

Ex:

```
System.out.print("Welcome \n to java");
```

output→Welcome
to java

Ex:

```
System.out.print("Ehab \t Qadoumi");
```

output→ Ehab Qadoumi

Ex:

```
System.out.print("special escape in  \" Java \" ");
```

output→special escape in " Java "

Ex:

```
System.out.print("aaaa\rbb");
```

output→ bbaa

Ex:

```
System.out.print("aaaa\bbb");
```

output→ aaab

Memory Concept :

- Local Variables
- Field Variables

Name	Description	Size*	Range*
char	Character or small integer.	1byte	signed: -128 to 127 unsigned: 0 to 255
short int (short)	Short Integer.	2bytes	signed: -32768 to 32767 unsigned: 0 to 65535
int	Integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
long int (long)	Long integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
bool	Boolean value. It can take one of two values: true or false	1byte	true or false
float	Floating point number.	4bytes	+/- 3.4e +/- 38 (~7 digits)
double	Double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)

Rules for naming variables:

- that starts with the letter A-Z, a-z and could start signal, $ _
- That does not start with a number.
- does not contain a blank space
- Do not be reserved names public, static. main, int, String ..
- Would be preferable to have a name, expressing what the object is doing Sum express collection Shi and so... .

Variables

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals, or characters in these variables.

There are two data types available in Java:

1. Primitive Data Types
2. Reference/Object Data Types

- **Primitive Data Types:**

There are eight primitive data types supported by Java. Primitive data types are predefined by the language and named by a key word. Let us now look into detail about the eight primitive data types. (int , double , float , short ,long , char , Boolean , byte) .

- **Reference Data Types:**

Reference variables are created using defined constructors of the classes. They are used to access objects. These variables are declared to be of a specific type that cannot be changed and the Default value of any reference variable is null.

In Java, all variables must be declared before they can be used. The basic form of a variable declaration is shown here:

Type identifier [= value][, identifier [= value] ...] ;

Examples :
int x ;
double y ;
int a,b,c ;
int a1=10 , a2 = 15 ;

☒ Java Types and Type Conversion (Casting)

Type conversion is the process of converting the data type of a variable for the duration of a specific operation. The standard form for a narrowing conversion is called a cast; it may risk your data , sometimes loss of precision

Ex:
```
int x ;
double  y= 3.0;
x = y ;              // Error : possible loss of precision
```

☒ There are more than way to conversion from type to another :

- **Implicit casting :**

There are times when a cast is performed implicitly by the compiler.

Ex:
```
if (3 > 'a')
{
...
}
```

In this case, the value of 'a' is converted to an integer value (the ASCII value of the letter a) before it is compared with the number 3.

- **Explicit conversion**

  ➔ **Syntax for a widening cast is simple:**

*NameOfOldValue = (**new type**) NameOfNewValue ;*

→This arrangement by numbering data type from smallest to largest :

Byte → short → int → long → float → double

If you go near arrows to then you do not need to convert , and his process called " **Promotion**"
Either if you go in the opposite, you need to convert "**Casting** "

Ex :
```
Int a = 5 ;
Long b = a ;  // I need convert int to double  → that's Ok …
```

Ex :
```
Long b = 5 ;
Int a = b ;  // I need convert long to int → need casting
          → int a = (int) b ;
```

Ex :

```
float c = 2.2 ;        // an floating point number store in java as double
                       // to convert there are two ways :
            → float c = (float)2.2 ;
            → float c = 2.2f ;
```

Ex :

```
double a = 11.2 ;
int x ;
x = a ;                // I need store double in integer → need casting
            →x = (int) a ;
```

> **Note that :** decimals are rounded by default Be sure you thoroughly understand the syntax for the types you want to cast; this process can get messy and loss of precision .

→This arrangement by lettering data type :

char → int → long

In this case, the value of letter is converted to an integer value (the ASCII value of the letter) before it is compared with the number .

Ex:

```
char a = 'A';          //that Ok …
```

Ex:

```
char a = 65;           //convert by ASCII to char
            → a=A
```

Ex:

```
int  l= 5 ;
char c = l ;           //convert int to char → need casting
            →char c = (char) l;
```

☒ Input from User :

A Scanner object can parse user input entered on the console or from a file. A Scanner breaks its input into separate tokens (which are typically separated by white space), and then returns them one at time.

```
import java.util.Scanner ;

public class CLASSNAME
{
        public static void main (String [] arg)
        {

                Scanner input = new Scanner (System.in);
                VariableName = input .XXXX();

        } // end main method

} // end of public class
```

Whenever using scanners, be sure to include the proper import line:
Import java.util.Scanner;
We will create scanners :Scanner input = new

System.in is : an InputStream which is typically connected to keyboard input of console programs.

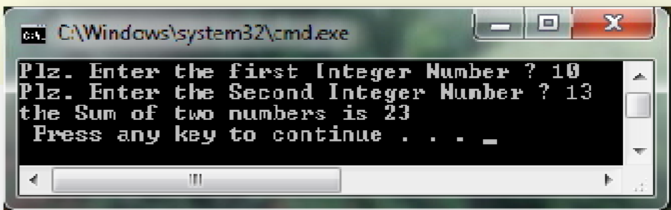
The Scanner provides methods to convert the tokens into values of different types.

- **nextInt() ;**→reads and converts next token to a integer value
- **nextDouble();**→reads and converts next token to a double value
- **next() ;**→reads next token and returns it as a String
- **nextLine();**→reads until the next new line and returns a String
- **nextBoolean() ;** → reads and converts next token to a Boolean value

Example : Write a program to inset 2 numbers from user and print there sum

```
import java.util.Scanner ;
public class TEST
{
        public static void main (String [] arg )
        {
          Scanner input = new Scanner (System.in);
          int num1 , num2 ;
          System.out.print("Plz. Enter the first Integer Number → ");
          num1 = input.nextInt();
          System.out.print("Plz. Enter the Second Integer Number → ");
          num2 = input.nextInt();

          System.out.printf("the Sum of two numbers is %d \n " , (num1+num2));
        }
}
```



☒ operations in java :

Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups:

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators
- Misc Operators

○ The Arithmetic Operators:

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The following table lists the arithmetic operators:

Operator	Description	Example
+	Addition - Adds values on either side of the operator	A + B will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	A - B will give -10
*	Multiplication - Multiplies values on either side of the operator	A * B will give 200
/	Division - Divides left hand operand by right hand operand	B / A will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	B % A will give 0
++	Increment - Increase the value of operand by 1	B++ gives 21
--	Decrement - Decrease the value of operand by 1	B-- gives 19

Priorities of Arithmetic operations :

1. What's inside the parentheses()
2. exponent
3. regular multiplication and division in the same priority *, / and%
4. addition and subtraction with the same priority + and-
5. In the case of equal priorities Start from left to right

- ### The Relational Operators:

There are following relational operators supported by Java languageAssume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

- ### The Bitwise Operators:

Java defines several bitwise operators which can be applied to the integer types, long, int, short, char, and byte.

➔ assume that a = 0011 1100 and b = 0000 1101

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
|	Binary OR Operator copies a bit if it exists in eather operand.	(A | B) will give 61 which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~	Binary Ones Complement Operator is unary and has the efect of 'flipping' bits.	(~A) will give -60 which is 1100 0011
<<	Binary Left Shift Operator. The left operands value s moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111
>>>	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

○ The Logical Operators:

The following table lists the logical operators:

Assume Boolean variables A holds true and variable B holds false then:

<table>
<tr><th>Operator</th><th>Description</th><th>Example</th></tr>
<tr><td>&&</td><td>Called Logical AND operator. If both the operands are non zero then then condition becomes true.</td><td>(A && B) is false.</td></tr>
<tr><td>||</td><td>Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.</td><td>(A || B) is true.</td></tr>
<tr><td>!</td><td>Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.</td><td>!(A && B) is true.</td></tr>
</table>

○ The Assignment Operators:

<table>
<tr><th>Operator</th><th>Description</th><th>Example</th></tr>
<tr><td>=</td><td>Simple assignment operator, Assigns values from right side operands to left side operand</td><td>C = A + B will assigne value of A + B into C</td></tr>
<tr><td>+=</td><td>Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand</td><td>C += A is equivalent to C = C + A</td></tr>
<tr><td>-=</td><td>Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand</td><td>C -= A is equivalent to C = C - A</td></tr>
<tr><td>*=</td><td>Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand</td><td>C *= A is equivalent to C = C * A</td></tr>
<tr><td>/=</td><td>Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand</td><td>C /= A is equivalent to C = C / A</td></tr>
<tr><td>%=</td><td>Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand</td><td>C %= A is equivalent to C = C % A</td></tr>
<tr><td><<=</td><td>Left shift AND assignment operator</td><td>C <<= 2 is same as C = C << 2</td></tr>
<tr><td>>>=</td><td>Right shift AND assignment operator</td><td>C >>= 2 is same as C = C >> 2</td></tr>
<tr><td>&=</td><td>Bitwise AND assignment operator</td><td>C &= 2 is same as C = C & 2</td></tr>
<tr><td>^=</td><td>bitwise exclusive OR and assignment operator</td><td>C ^= 2 is same as C = C ^ 2</td></tr>
<tr><td>|=</td><td>bitwise inclusive OR and assignment operator</td><td>C |= 2 s same as C = C | 2</td></tr>
</table>

○ Misc Operators :

There are few other operators supported by Java Language.

▪ Conditional Operator (? :)

Conditional operator is also known as the ternary operator. This operator consists of three operands and is used to evaluate boolean expressions. The goal of the operator is to decide which value should be assigned to the variable. The operator is written as :

➔ **Syntax :**

```
Variable x = (expression)? Value if true: value if false
```

☒ Increment and Decrement Operator :

The ++ and the-- are java's increment and decrement operators, the increment operator increases its operand by one. The decrement operator decreases its operand by one. For example, this statement:

x = x + 1;

can be rewritten like this by use for the increment operator:

```
x++;        // done process then increasing the value
```

Similarly, this statement:

x = x –1;

is equivalent to

```
x--;        // done process then decreasing  the value
```

Example :

```
int x = 3 ;
System.out.println("x = "+ x );        //3
x++ ;                                  //4
System.out.println("x = "+ x );        //x=4
System.out.println("x = "+ +x );       //x =5
System.out.println("x = "+ x );        //x =5
System.out.println("x = "+ x-- );      //x =5  and decrement x to be 4
System.out.println("x = "+ x++ );      //x =4 and increment x to be 5
System.out.println("x = "+ --x );      //x = 4
```

☒ Input and Output Dialog boxes

In non-swing application we were using **System.in** class for input or output some text or numeric values but now in the swing application we can use **JOptionPane** to show the output or show the message. This way of inputting or outputting works very efficiently in the Swing Applications. The window for showing message for input or output makes your application very innovative.

JOptionPane class is available in the **javax.swing.*;** package. This class provide various **types of dialog** box as follows:

1. A simple **message** dialog box **which has only one button** i.e. "Ok". This type of message dialog box is used only for showing the appropriate message and user can finish the message dialog box by clicking the "Ok" button.
2. A **message** dialog box **which has two or three buttons**. You can set several values for viewing several message dialog box as follows:
   1.) "Yes" and "No"
   2.) "Yes", "No" and "Cancel"
   3.) "Ok", and "Cancel"
3. A input dialog box which contains two buttons "Ok" and "Cancel".

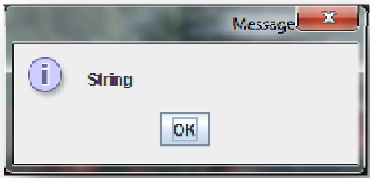
The **JOptionPane** class has three **methods** as follows:

- **showMessageDialog():** First is the **showMessageDialog()** method which is used to display a simple message.
- **showInputDialog():** Second is the **showInputDialog()** method which is used to display a prompt for inputting. This method returns a String value which is entered by you.
- **showConfirmDialog():** the last or third method is the **showConfirmDialog()** which asks the user for confirmation (Yes/No) by displaying message.

○ showMessageDialog():

This method is used to show a message dialog box which contains some text messages. This is being used with two arguments in the program where the first argument is the parent object in which the dialog box opens and another is the message which has to be shown.

```
JOptionPane.showMessageDialog(null,"String");
```

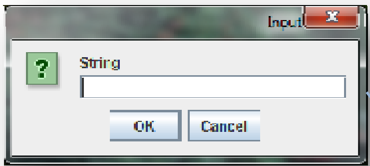


it's a **screen to display** the results only, you do not need to variable to store the return value .

○ showInputDialog():

This method is used to show a input dialog box which contains some text messages and text area to input the text in her. This is being used with one argument in the program where you enter the value you want in her and store in variable .

```
String Var.Name = JOptionPane.showInputDialog("String");
```



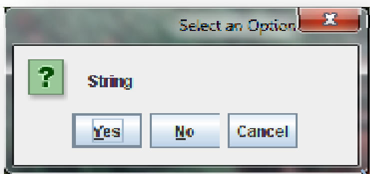
it's a data **entry screen**, you need to put a variable to store the entered value.

Variable must be of type String where this method return String

○ showConfirmDialog():

And the last or third method is the **showConfirmDialog()** which asks the user for confirmation (Yes /No) by displaying message. This method return a numeric value either 0 or 1, If you click on the "Yes" button then the method returns 1 otherwise 0.

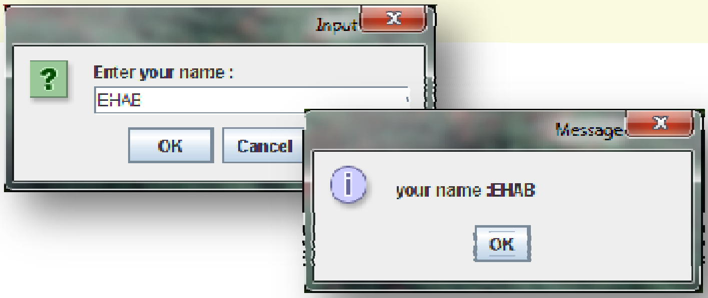
```
int Var.Name  = JOptionPane.showConfirmDialog(null,"String");
```



It's a screen asks the user for confirmation , return 1 if click yes or 0 if no … so you will put it in an integer variable .

Example : I need to Enter my name and show in MessageBox

```
import javax.swing.JOptionPane ;
public class MESSAGE
{
        public static void main(String [] arg)
        {
        String Name = JOptionPane.showInputDialog("Enter your name :");
        JOptionPane.showMessageDialog(null,"your name :" + Name);
        }
}
```



We said earlier that showInputDialog method return an String value !

If you want to enter numbers and perform calculations ?

➔ there are many methods to parsing from String to numbers

☒ Parsing method :

- Integer.parseInt()
- Double.parseDouble()
- Float.parseFloat()

➔ **Syntax Parsing :**

*DataType Var.Name = **XXX.parseXXX**(var_you_need_parsing);*

Example : write a program in java that will inert the name of student and the first , mid and final Exam , then calculation Total of marks .

Note : use input and output Dialog Boxes !

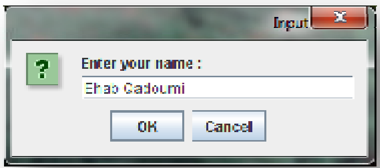
```
import javax.swing.* ;
public class Example
{
	public static void main (String [] arg)
	{
	String Name , F_mark , M_mark , Final_mark ;

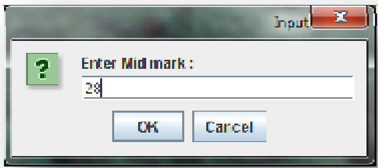
	Name = JOptionPane.showInputDialog("Enter your name :");
	F_mark = JOptionPane.showInputDialog("Enter the first mark :");
	M_mark = JOptionPane.showInputDialog("Enter Mid mark :");
	Final_mark = JOptionPane.showInputDialog("Enter final mark :");

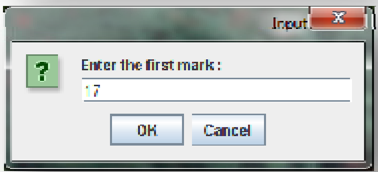
	double x  , y , z ;
	x = Double.parseDouble(F_mark);
	y = Double.parseDouble(M_mark);
	z = Double.parseDouble(Final_mark);

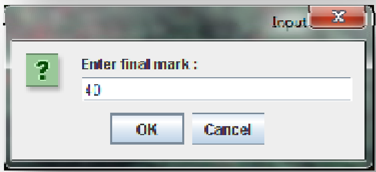
	JOptionPane.showMessageDialog(null , "your name : " + Name + " \n
your mark : " + (x+y+z));

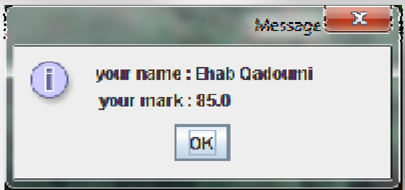
	}
}
```











Control Structures

Control statements are used in programming languages to cause the flow of control to advance and branch based on changes to the state of a program.

In Java, **control statements** can be divided under the following three **categories**:

- Sequence.
- Selection.
- Looping.

☒ Selection Statement :

Selection statements are used in a program to choose different paths of execution based upon the outcome of an expression or the state of a variable.

- **if single-selection statement**
- **if –else double selection statement**
- **nested if- else statement**

○ if single-selection statement :

The if statement executes a block of code only if the specified expression is true. If the value is false, then the if block is skipped and execution continues with the rest of the program. You can either have a single statement or a block of code within an if statement. Note that the conditional expression must be a Boolean expression.

→if statement has the following syntax:

if (conditional expression)
Statementaction;

Example :

```
public class IfStatementDemo
{
        public static void main(String[]args)
        {
                int a = 10, b = 20;
                if (a > b)
                        System.out.println("a > b");
                if (a < b)
                        System.out.println("b > a");
        }
}
```

Output→b > a

- ### if –else double selection statement :

The if/else statement is an extension of the if statement. If the statements in the if statement fails, the statements in the else block are executed. You can either have a single statement or a block of code within if-else blocks. Note that the conditional expression must be a Boolean expression.

→If-else statement has the following syntax:

```
if (conditional expression)
        statement action ;
else
        statement action ;
```

Example:

```
public class IfElseStatementDemo {

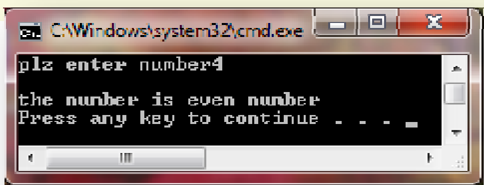
        public static void main(String[] args) {
                int a = 10, b = 20;
                if (a > b) {
                        System.out.println("a > b");
                } else {
                        System.out.println("b > a");
                }}}
```

Output→ b > a

Example: write a program that enter a number and check if even or odd.

```
import java.util.Scanner;
public class NUMBER {
        public static void main(String args[])          {
        Scanner inp=new Scanner (System.in);
        String s;
        int y;
        System.out.print("plz enter number");
        y=inp.nextInt();
        System.out.println();

        if(y%2==0)
                System.out.println("the number is even number");
        else
                System.out.println("the number is odd number");
        }}
```



- ### nested if- else statement :

Based on the same principle accepted by the sentence, but there is one difference that there is here more than the condition . In the absence of the first condition is met, it will move to the second condition to achieve the condition come out of the sentence .

→Nested if-else statement syntax:

if (conditional expression 1)
*statement action **1 ;***
else if (conditional expression 2)
*statement action **2;***

:

else if (conditional expression n)
*statement action **n*** ;

Example :

```
public class NestedIfElseStatementDemo
{
        public static void main(String[] args)
        {
                int a = 10, b = 10;
                if (a > b)
                        System.out.println("a > b");
                else if (b>a)
                        System.out.println("b > a");
                else
                        System.out.println("Equal");

        }
}
```

Output→ Equal

Example :

```
public class NestedIfElseStatementDemo2
{
		public static void main(String[] args)
		{
		int grade = 88 ;
		if (grade >= 0 && grade <= 100)
		{

		if (grade >= 90)
				System.out.print("A");
		else if (grade >=80)
				System.out.print("B");
		else if (grade >=70)
				System.out.print("C");
		else if (grade >=60)
				System.out.print("D");
		else
				System.out.print("F");

		} //end outer if
		} //end main method
}//end public class
```

Output→ B

☒ Looping statements :

Sometimes, you want to repeat something many times, you need to use the repeat statement.

- **while statement**
- **do – while statement**
- **for statement**
- **nested for**
- **switch case**

○ while statement :

The while statement is a looping construct control statement that executes a block of code while a condition is true. You can either have a single statement or a block of code within the while loop. The loop will never be executed if the testing expression evaluates to false. The loop condition must be a Boolean expression.

→The syntax of the while loop is:

```
initialization;

while (loop condition)

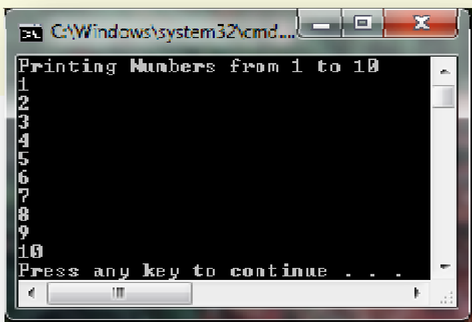
{
statements;

counter increment / decrement ;

}
```

Example: write a program that print numbers 1-10

```
public class WhileLoopDemo
 {
            public static void main(String[] args)
            {
                        int count = 1;
                        System.out.println("Printing Numbers from 1 to 10");
                        while (count <= 10)
                        {
                                    System.out.println(count++);
                        }
            }
}
```



- **do – whil statement :**

The do-while loop is similar to the while loop, except that the test is performed at the end of the loop instead of at the beginning. This ensures that the loop will be executed at least once. A do-while loop begins with the keyword do, followed by the statements that make up the body of the loop. Finally, the keyword while and the test expression completes the do-while loop. When the loop condition becomes false, the loop is terminated and execution continues with the statement immediately following the loop. You can either have a single statement or a block of code within the do-while loop.

→The syntax of the do-while loop is:

```
initialization ;

do
{

statements ;

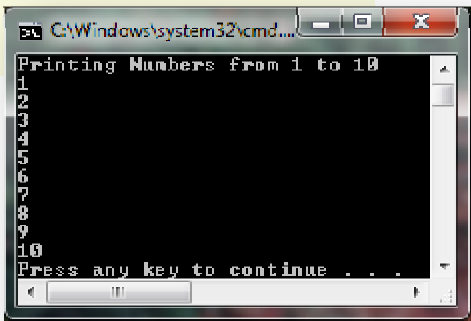
counter increment / decrement ;

}
while (loop condition);
```

Below is an **example** that demonstrates the looping construct namely do-while loop used to print numbers from 1 to 10 :

```
public class DoWhileLoopDemo
{

        public static void main(String[] args)
         {
                int count = 1;
                System.out.println("Printing Numbers from 1 to 10");
                do {
                        System.out.println(count++);
                } while (count <= 10);
        }
}
```



- ### For Loops :

The for loop is a looping construct which can execute a set of instructions a specified number of times. It's a counter controlled loop.

→The syntax of the loop is as follows:

for *(initialization ; loop condition ; increment expression)*

```
{
        Statements;
}
```

The first part of a for statement is a starting **initialization**, which executes once before the loop begins. The initializationsection can also be a comma-separated list of expression statements. The second part of a for statement is a test **expression** (condition). As long as the expression is true, the loop will continue. If this expression is evaluated as false the first time, the loop will never be executed. The third part of the for statement is the **body of the loop**. These are the instructions that are repeated each time the program executes the loop. The final part of the for statement is an **increment expression** that automatically executes after each repetition of the loop body.

Typically, this statement changes the value of the counter, which is then tested to see if the loop should continue. All the sections in the for-header are optional. Any one of them can be left empty, but the two semicolons are mandatory. In particular, leaving out the **loop condition** signifies that the loop condition is true. The (; ;) form of for loop is commonly used to construct an infinite loop ...

Statement is executed after the loop body is done. Generally it is being used to increment or decrement the loop variable.

Following **example** shows use of simple for loop.

```
for ( int i = 0 ; i < 3 ; i++)
{
          System.out.print("i is : " + i + "     ");
}
```

Output → i is : 0   i is :1   i is :2

- It is possible to initialize multiple variable in the initialization block of the for loop by separating it by comma as given in the below **example**.

```
for ( int i=0 , j=5  ;  i<5  ;  i++)
```

- It is also possible to have more than one increment or decrement section as well as given below.

```
for ( int i=0 ; i <5 ; i++ , j++)
```

- If you put a semicolon at the end of the sentence or that he was the starting value is on the rise and the condition is true, but the amount of increase is declining, it will not be implemented and there will be no output.

```
for ( int i=0 ; i <5 ; i++ ) ;
```

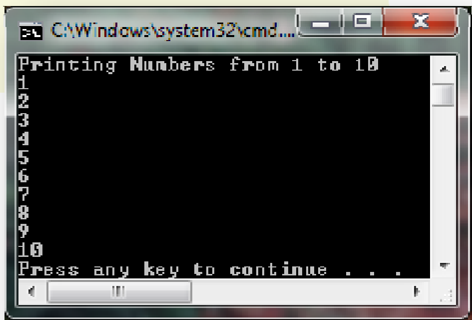
However it is not possible to include declaration and initialization in the initialization block of the for loop.

Also, having multiple conditions separated by comma also generates the compiler error. However, we can include multiple condition with && and || logical operators.

Below is an **example** that demonstrates the looping construct namely for loop used to print numbers from 1 to 10.

```
public class ForLoopDemo
{

		public static void main(String[] args)
		 {
				System.out.println("Printing Numbers from 1 to 10");
				for (int count = 1; count <= 10; count++)
				{
						System.out.println(count);
				}
		}
}
```



Example : write a program that ask user to insert end value and print an even numbers from 0 to end value .

```
import java.util.Scanner ;
public class EVEN_NUMBERS
{
        public static void main(String[] args)
        {
                Scanner input = new Scanner (System.in);

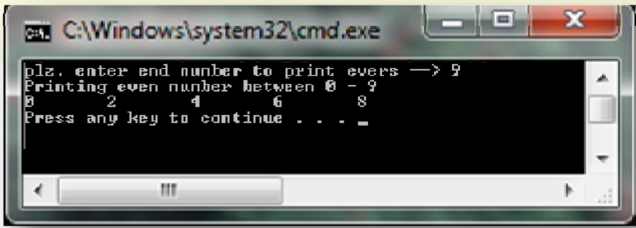
                System.out.print("plz. enter end number to print evers --> ");

                int end = input.nextInt(); // input from user

        System.out.println("Printing even number between 0 - " + end );

                for (int i= 0 ; i <= end ; i++)
                {
                        if (i % 2 ==0) // if even print it
                        System.out.print(i + "\t");
                } // end if

                System.out.println();
        } //end main method
} //end of class
```



- **nested for :**

he placing of one loop inside the body of another loop is called **nesting**. When you "**nest**" two loops, the outer loop takes control of the number of complete repetitions of the inner loop. While all types of loops may be nested, the most commonly nested loops are **for** loops.

→The syntax of the nested for loop is as follows:

```
for (initialization 1; condition 1 ; counter 1 )
{
        for (initialization 2; condition 2 ; counter 2)
        {
        statements ;
        }
}
```

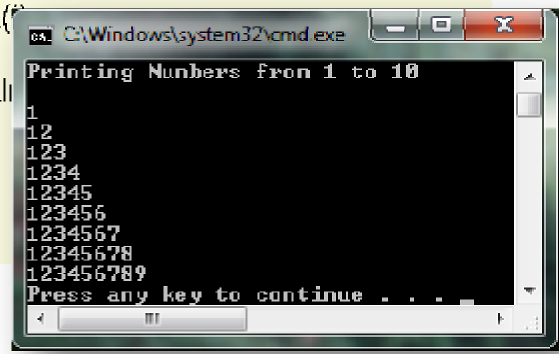
Example *:*

```
public class ForLoopDemo {

        public static void main(String[] args)
         {
                System.out.println("Printing Numbers from 1 to 10");
                for (int i = 1; i <= 10; i++)
                {
                for ( int j = 1 ; j <i ; j++)
                        System.out.print(

                        System.out.printl

                }
        }
}
```

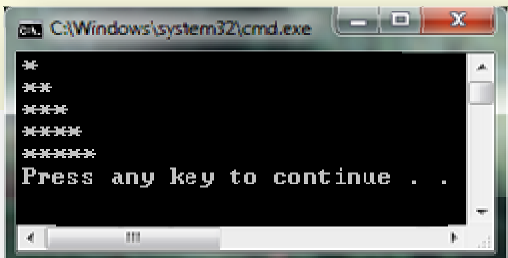


Example :

```
import java.util.Scanner ;
public class Stars
{
        public static void main(String[] args)
        {
        for (int i=1 ; i <= 5 ; i++)
        {
                for(int j = 1 ; j <= i ; j++)
                System.out.print("*") ;

                System.out.println();
        } //end for

        } //end main method
} //end class
```



Example :

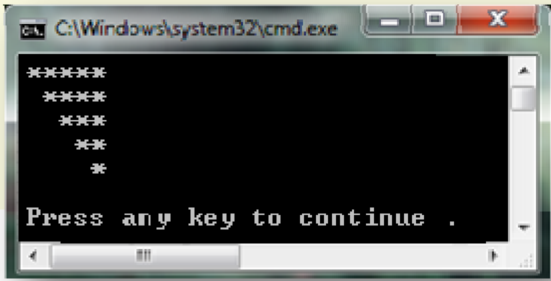
```
import java.util.Scanner ;
public class INVERSE_STARS{
        public static void main(String[] args) {
        int i , s , j ;

        for (i=5 ; i >0 ; i--)
        {
                for( j = 1 ; j <= i ; j++)
                System.out.print("*") ;

                System.out.println();

        for(s=5 ; s>=j-1 ; s--)
        System.out.print(" ");
        }

   System.out.println();   }}
```



Example :

```
Public class DYMOND{
public static void main(String[] args){
int x=11;
int y=x/2; // spaces
int z=1; // *
boolean b1=true;
boolean b2= true;

        for(int i=0;i<x;i++)
        {
                for(int j=0;j<y;j++)
                {
                System.out.print(" ");
                }

                for(int k=0 ; k<z ; k++)
                {
                System.out.print("*");
                }

                if(y==0) b1=false;
                if(z==x) b2=false;
                y=b1?y-1:y+1;
                z=b2?z+2:z-2;

        System.out.println();
        }  }}
```



- **switch case :**

A *switch* statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

→The syntax of switch case is follows:

```
switch (expression)

{

case value 1 :  statement 1 ;

break;   //optional

case value 2 :  statement 2 ;

break;   //optional

:

case value n1 :  statement n ;

break;   //optional

default:   //optional

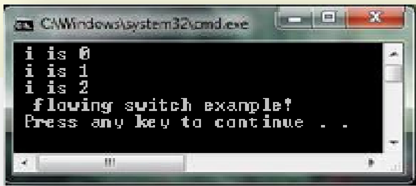
statement;              }     //end of switch
```

The following rules apply to a switch statement:

- The variable used in a switch statement can only be a byte, short, int, or char.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- The value for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.
- When the variable being switched on is equal to a case, the statements following that case will execute until a *break* statement is reached.
- When a *break* statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- Not every case needs to contain a break. If no break appears, the flow of control will *fall through* to subsequent cases until a break is reached.
- A *switch* statement can have an optional default case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No break is needed in the default case.

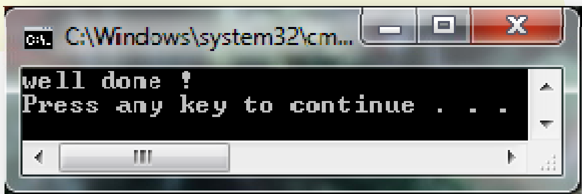
Example:

```
public class SwitchExample
{
        public static void main(String[] args)
        {
        int i=0;
        switch(i)
        {
        case 0:
        System.out.println("i is 0");
        case 1:
        System.out.println("i is 1");
        case 2:
        System.out.println("i is 2");
        default:
        System.out.println(" flowing switch example!");
        } //end switch bodt
        } //end main method
} // end class
```



Example:

```
public class Test
{
        public static void main (String [] arg)
        {
                char grade = 'B';
                switch (grade)
                {
                case 'A' :
                System.out.println("Excelebt !");
                break ;
                case 'B' : case 'C' :
                System.out.println("well done !");
                break ;
                case 'D' :
                System.out.println("you pass !");
                break ;
                default :
                System.out.println("Invalid grade !");
                } //end switch
        }//end main method
} //end public class
```



☒ break and continue in Java :

The statements break and continue in Java alter the normal control flow of compound statements. The break and continue statements do not make sense by themselves. Without labels, break and continue refer to the most closely enclosing **for, while, do, or switch** statement. With labels the break statement is legal in a labeled if statement or a labeled {} block.

The **break** keyword is used to stop the entire loop. The break keyword must be used inside any loop or a switch statement, it will stop the execution of the innermost loop and start executing the next line of code after the block.

The **continue** keyword can be used in any of the loop control structures. It causes the loop to immediately jump to the next iteration of the loop.

- In a for loop, the continue keyword causes flow of control to immediately jump to the update statement.
- In a while loop or do/while loop, flow of control immediately jumps to the Boolean expression.

Example :

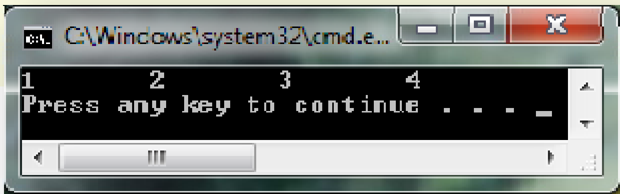
```
public class B_K
{
        public static void main (String [] arg)
        {
                for (inti=1 ; i<10 ; i++)
                {
                If (i==5) break ; // when i == 5 >> stop

                System.out.print(i + "\t");
                } //end for

        System.out.println();

        } //end main method

} //end public class
```



Classes and Objects

☒ Class and Objects :

Class Definition: A class is a template for an object, and defines the data fields and methods of the object. The class methods provide access to manipulate the data fields.

➔ The general form of a simple class is:

```
modifier class classname
{
        modifier data-type instance-variable1;
        modifier data-type instance-variable2;
        ...
        modifier data-type instance-variableN;


        modifier return-data-type class-method1( parameters )
        {
        statements;
        }
        modifier return-data-type class-method2( parameters )
        {
        statements;
        }
        ...
        modifier return-data-type class-method2( parameters )
        {
        statements;
        }

}
```

Notes :

- ➔ All **fields** and **methods** of a class are always available to the code in the class itself; the class always "knows" about its own data fields and methods.

- ➔ At least, in your **code** it have one class "Public class" and one method "main method".

☒ Creating Objects(new Operator) :

The new operator creates a single instance of a class and returns a reference to that object. During execution of a new operator, Java allocates enough space to store the fields of the object. When initialization of the object is complete, Java returns a reference to the new object. If insufficient resources (memory space) are available to create the object, then the system may run the garbage collector.

➔ The general form of declare an object:

```
ClassName ObjectName = new ClassName( Parameter );
```

- **ClassName** : the name of public class
- **ObjectName** : any valid name such as : obj … .
- **new** : key word to call constructor " will talk with him later ".
- **Parameter** : if you need to sent values or anything , you can use it but it an optional .

❖ Method Declaration :

A method is defined inside of a class definition at the same level as the instance variables. Each method in a class may take on a specific number of input parameters. The method may return a data type, which may be a primitive data type or a reference type (object). Furthermore, a class method may be proceeded by a **modifier** to define the accessibility of the method, for example public or private…

➔ The general form of declare an method :

```
Modifier [static] return-type class-method( parameter-list )
{
method statements;
}
```

- class maybe contain more than one method , but at least one "main"
- **Modifier** : public , private , protected , Default .
  - ➔ **Public**: A public member function can be invoked by any other member function in any other object or class.
  - ➔ **Protected**: A protected member function can be invoked by any member function in the class in which it is defined or any subclasses

of that class; even if the subclass is in another package (In Java, protected members can be invoked in any class in the same package too).

➔ **Private**: A private member function can only be invoked by other member functions in the class in which it is defined, but not in the subclasses.

➔ **Default** (No keyword, simply leave it blank): The member function is effectively public to all other classes within the same package, but private to classes external to the package. This is sometimes called package visibility or friendly visibility.

- **static** : static methods use no instance variables of any object of the class they are defined in. If you define a method to be static, you will be given a rude message by the compiler if you try to access any instance variables. It's an optional **keyword** .

- Java does not support a variable number of parameters (as in C).

- If the method has no parameters, then the method declaration should include a pair of **empty parenthesis**.

☒ Constructor :

A class **constructor** can be invoked to assign specific values directly to the data fields.

- Constructors have the **same name as the class.**
- Constructors are not class methods and **cannot return** a data type.

➔ Syntax of constructor :

```
public class ClassName
{
        DataType Var.Name ;
        DataType Var.Name ;
        :
        DataType Var.Name ;

        public ClassName (parameter )
        {
        Var.Name  = value ;
        :
        Var.Name  = value ;
        }
}
```

remember :
Field variables Take default values when defined .
int→ 0
double→ 0.0
char→ ' '
String → null

The parameter in constructor is **optional** , if the constructor has not parameter that name **Defoult constructor** , and if has a parameter that name **with parameter**

Example : this example discussing default constructor .

```
public class Test
{
        static int x ;
        static double y ;

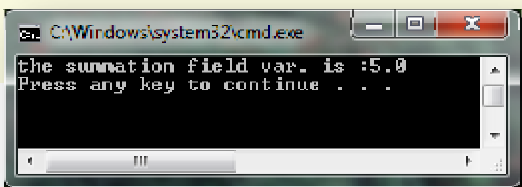
        public Test ()
        {
                x = 5 ;
        }

        public void sum ()
        {
                System.out.println("the summation field var. is :" + (x+y));
        }

        public static void main (String [] arg)
        {
                Test obj = new Test ();
                obj.sum();
        }
}
```

Declare default Constructor
publicClassName()

when you create an object ; the constructor automatically call …



Ehab A. Qad'oumi ...

Example : example discussing constructor with parameter .

```
public class Test1
{
        static int x ;
        static double y ;

        public Test1(int a , double b)
        {
                x = a ;
                y = b ;
        }

        public void sum ()
        {
                System.out.println("the summation field var. is :" + (x+y));
        }

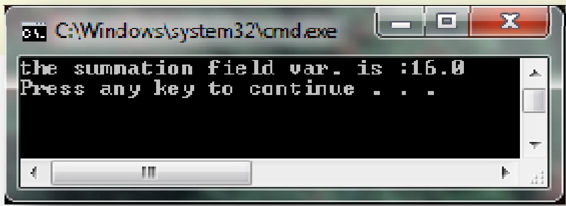
        public static void main (String [] arg)
        {
                int x = 7 ;
                double y=9 ;

                Test1 obj = new Test1 (x , y);
                obj.sum();
        }
}
```

Declare Constructor with parameter

Store the value of variable **a** in x
And
Store the value of variable **b** in y .

Calling the method :
ObjectName.MethodName (Parameter) ;



Example :

```
import java.util.Scanner ;
public class project
{
        String Name ;  //field variable

        public String set_name (String N)
        {
                Name = N;
                return Name ;
        }

        public void get_name ()
        {
                System.out.println("your name is :" + Name );
        }

        public static void main (String [] arg)
        {
                Scanner input = new Scanner (System.in);
                project obj = new project ();

                System.out.print("Plz. Enter your name ->");
                String My_name = input.nextLine();

                obj.set_name(My_name);

                obj.get_name ();

                System.out.println("Done set and get method ...");
        }

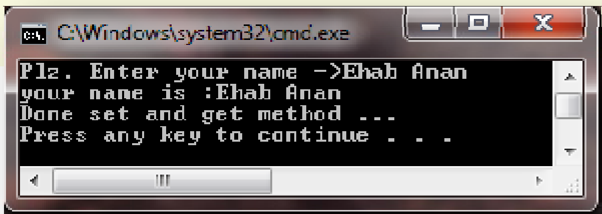
}
```

Method that return value , must be declared with return value data type such that int , String and put return Keyword .

Method without return value that must declared with void data type and no return Keyword .

When method declared with static , you don't need to do instance (declare object).

Call methods .
If the other method non-static , you will declare an object and call the methods by her .
Note: main method must be static.



⊠ Method Overloading

In Java, multiple methods with the same name can be defined in a class, provided that each method has a different parameter list. The order and type of the parameters in the list define the signature of the method.

```
public void m1();              → m1()
public void m1(int );          → m1(int)
public void m1(int , double);  → m1(int , double);
public void m1(double , int);  → m1(double , int);
```

Process called **Overloading** if there are more than Method with the same name but different number or type parameter

Example :

```
public class Circle
{
        static double x,y ;

        public void distance()
        {
        System.out.println(Math.sqrt(x*x + y*y) );
        }

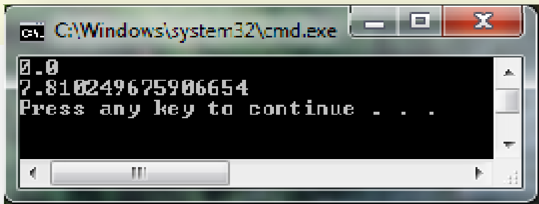
        public double distance(double xcordinate , double ycordinate)
        {
        double dx = x - xcordinate ;
        double dy = y - ycordinate ;
        return Math.sqrt(dx*dx + dy*dy);
        }

        public static void main (String [] arg)
        {
        Circle obj = new Circle () ;
        obj. distance ();
        System.out.println(obj.distance(5.0,6.0));
        }
}
```

method that does not return value .
void distance() → no parameter

method that return value .
double distance(double ,double)
→parameter

create an object



☒ Scope of Declarations :

You have seen declarations of various Java entities, such as classes, methods, variables and parameters. Declarations introduce names that can be used to refer to such Java entities. The scope of a declaration is the portion of the program that can refer to the declared entity by its name. Such an entity is said to be "in scope" for that portion of the program. This section introduces several important scope issues.

The basic scope rules are as follows:

- The scope of a parameter declaration is the body of the method in which the declaration appears.
- The scope of a local-variable declaration is from the point at which the declaration appears to the end of that block.
- The scope of a local-variable declaration that appears in the initialization section of a for statement's header is the body of the for statement and the other expressions in the header.
- The scope of a method or field of a class is the entire body of the class. This enables non-static methods of a class to use the class's fields and other methods.

Example:

```
public class SCOPE
{
double var1 ;
static int var2 ;

        public static void main(String [] arg)
        {
        int a = 5 ;

        for (int i = 1 ; i <10 ; i++)
        System.out.print(i++);

        }
}
```

The scope of field variable all of class !
But , **warning** : if you would use field variable in **static method** ; the **variable** must be declared with **static**

The scope of a local-variable declaration is from the point at which the declaration appears to the end of that block .

The scope of variable in statement is from the body of statement .

☒ The Math Class :

It is hard to avoid the Math class in any Java program that requires scientific or other numeric computations. As with the wrapper classes, the Math class is part of the **java.lang package**; so, methods may be used without an explicit import statement. Table 1 summarizes the most common methods in the Math class. For a full listing, see the Java API documentation. Unlike the wrapper classes, the Math class contains no constructor method: Math objects cannot be instantiated. Therefore, all methods in Table 1 are class methods (aka static methods).

Method	Purpose
`abs(double a)`	returns a `double` equal to the absolute value of a `double` value `a`
`asin(double a)`	returns a `double` equal to the arc sine of an angle `a`, in the range of -pi/2 through pi/2
`exp(double a)`	returns a `double` equal to the exponential number e (i.e., 2.718...) raised to the power of a `double` value `a`
`log(double a)`	returns a `double` equal to the natural logarithm (base e) of a `double` value `a`
`pow(double a, double b)`	returns a `double` equal to of value of the first argument `a` raised to the power of the second argument `b`
`random()`	returns a `double` equal to a random number greater than or equal to 0.0 and less than 1.0
`rint(double a)`	returns a `double` equal to the closest `long` to the argument `a`
`round(double a)`	returns a `long` equal to the closest `int` to the argument `a`
`sin(double a)`	returns a `double` equal to the trigonometric sine of an angle `a`
`sqrt(double a)`	returns a `double` equal to the square root of a `double` value `a`
`tan(double a)`	returns a `double` equal to the trigonometric tangent of an angle `a`
`toDegrees(double a)`	returns a `double` equal to an angle measured in degrees equal to the equivalent angle measured in radians `a`
`toRadians(double a)`	returns a `double` equal to an angle measured in radians equal to the equivalent angle measured in degrees `a`
`E` `double`	holds the value of e, the base of natural logarithms, accurate to about 15 digits of precision
`PI` `double`	holds the value of pi, the ratio of the circumference of a circle to its radius, accurate to about 15 digits of precision
max `(double a, double b)`	**return the maximum between two numbers**
min `(double a, double b)`	**return the minimum between two numbers**

Ex:
System.out.print(Math.pow(2,3)) ; → 8

Ex:
System.out.print(Math.pow(25.0, 0.5)); → 5.0

Ex:
System.out.print(Math.abs(-3)) ; → 3

Ex:
System.out.print(Math.random()); →Ran. # between 0-1

Ex:
System.out.print(Math.PI); → 3.1415

Ex:
System.out.print(Math.max(30,50)); → 50

Ex:
System.out.print(Math.max(10, Math.max(5,15))); → 15

Ex:
System.out.print(Math.min(30,50)); → 30

Ex:
System.out.print(Math.min(5, Math.max(7,3))); → 5

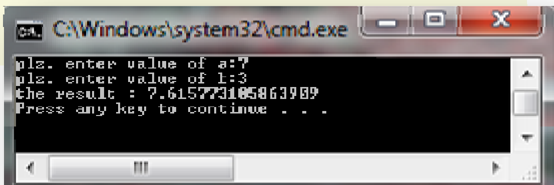
Example : write a program to solutioon this **expressions :** $\frac{\sqrt{a^2+b^2}}{5}$

```
import java.util.Scanner ;
public class expressions
{
            public static void main(String[] args)
            {
            Scanner input = new Scanner (System.in );

            int a ,b ;
            double result;
            System.out.print("plz. enter value of a:");
            a= input.nextInt();
            System.out.print("plz. enter value of b:");
            b= input.nextInt();

            result = Math.sqrt(Math.pow(a,2) + Math.pow(b,2)) ;

            System.out.println("the result : " + result);
            } //end main method
} // end class
```



Ehab A. Qad'oumi ...

☒ Random Numbers :

Java has a rich toolkit for generating random numbers, in a **class** named "**Random**". This document is a quick guide to using Random. Random can generate many kinds of random number

☒ Gaining Access to Random

Random is defined in the "**java.util**" library package, so any Java source file that uses Random must begin with a line of the form

```
import java.util.Random;

    or

import java.util.*;
```

☒ Creating Random Number Generators

The easiest way to initialize a random number generator is to use the parameterless constructor, for example

*Random R = **new** Random();*

However, in using this constructor you should recognize that algorithmic random number generators are not truly random, they are really algorithms that generate a fixed but random-looking sequence of numbers.

☒ Generating Random Integers

To generate a random integer from a Random object, send the object a "nextInt" message. This message takes no parameters, and returns the next integer in the generator's random sequence. Any Java integer, positive or negative, may be returned. Integers returned by this message are uniformly distributed over the range of Java integers. Here is an example, assuming that "generator" is an instance of Random:

```
int r = generator.nextInt();
```

Often, programmers want to generate **random integers between 0 and some upper bound**. For example, perhaps you want to randomly pick an index into an array of *n* elements. Indices to this array, in Java, range from 0 to*n*-1. There is a variation on the "**nextInt**" message that makes it easy to do this: If you provide an integer parameter to "nextInt", it will return an integer from a uniform distribution between 0 and one less than the parameter. For example, here is how you could use a random number generator object to generate the random array index suggested a minute ago:

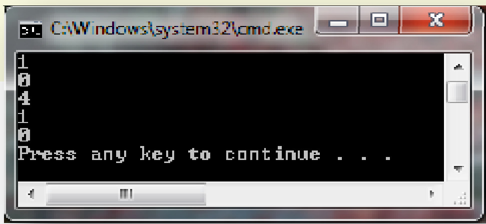
```
int randomIndex = generator.nextInt( n );
```

Example : write a program that print 5 random numbers

```
import java.util.Random ;
public class Ran {
          public static void main(String [] arg)
          {
                    Random R = new Random ();

                    for (int i = 0 ;i < 5 ; i++)
                    {
                              System.out.println(R.nextInt(5));
                    } //end for
          } // end main method
} //end class
```

number 5 that mean ,
Means that the number of possibilities is 5 . (0-4)



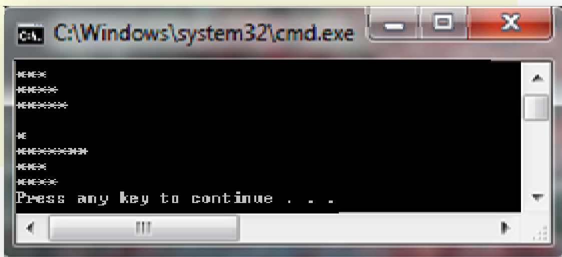
Example :

Write in Java program that prints a random number of stars more than 10 times each group on the line

```
import java.util.Random ;
public class expressions
{
          public static void main(String[] args){
          Random R = new Random () ;

          for (int i = 1 ; i < 10 ; i++)
          {
          int x = R.nextInt(10);

          for (int j = 1 ; j < x ; j++)
          System.out.print("*");
          System.out.println();
          }                    }}
```



Example : write a program that insert 10 numbers from user and check if the user entered is maches with an random number between 0-4 , in the end print number of correct gessing .

```
 import java.util.Random ;
import java.util.Scanner ;
public class Ran2
{
        public static void main(String [] arg)
        {
                Random R = new Random ();
                Scanner input = new Scanner (System.in);

                int x ;
                int counter = 0 ;

                System.out.println("System.out.println()");

                for (int i = 0 ;i < 10 ; i++)
                {
                System.out.print("Plz. Enter a number to check --> ");
                x = input .nextInt();

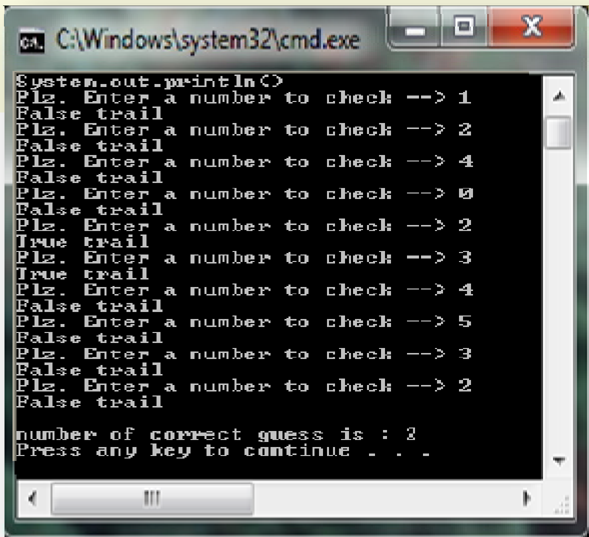
                if (x == R.nextInt(5))
                {
                        counter ++ ;
                        System.out.println("True trail ");
                }
                else
                        System.out.println("False trail ");
                }

                System.out.println();
                System.out.println("number of correct guess is :" + counter );

        } //end main metnod

} //end class
```

if the user enterd maches with random number increesing variable ny 1



☒ Java Final Keyword :

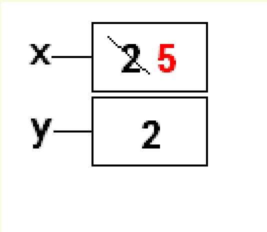
The **final keyword** has more than one meaning :

- ➔ a final class cannot be extended
- ➔ a final method cannot be overridden
- ➔ final fields, parameters, and local variables **cannot change** their value once set .

Example :

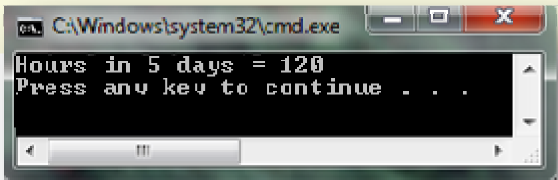
```
public class Test_Final
{
            public static void main (String [] arg)
            {
            int x = 2 ;
            System.out.print ("x = " + x ); // x= 2
            x = 5 ;
            System.out.print ("x = " + x ); // x=5

            final int y = 2 ;
            System.out.print ("y = " + y); // y= 2
             y = 5 ;              // Error : cannot assign a value to final variable y
            System.out.print ("y = " + y);
          }
}
```



Example :

```
public class FinalVariableExample
{
            public static void main(String[] args)
            {
            final int hoursInDay=24;
            System.out.println("Hours in 5 days = " + hoursInDay * 5);
            }
}
```



☒ This class :

Within an instance method or a constructor, this is a **reference to the current object** , the object whose method or constructor is being called. You can refer to any member of the current object from within an instance method or a constructor by using this .

Example :

```
 public class Field_var_use
{
            int x ;
            double y ;

            public Field_var_use () //Defoult constructor
            {
            int x = 10 ;
            System.out.println("x in local is : " + x );
            System.out.println("x in field is : " + this.x);
            }

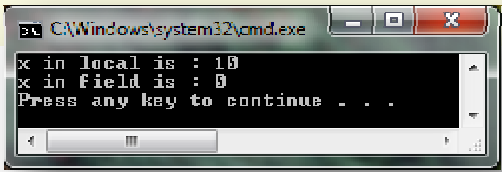
            public static void main(String [] arg)
            {
            Field_var_use obj = new Field_var_use () ;
            } //end main method

} // end public class
```

Field variables that initially vales by default :
int → 0
double → 0.0
String → null
char → ''

this → to call field var. if there is an local variable found that carry the same name !

when you declare an object , constructor will be automatically calling



Arrays

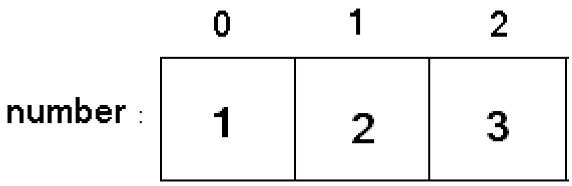
Suppose we have here three variables of type int with different identifiers for each variable.

int number1;
int number2;
int number3;
number1 = 1;
number2 = 2;
number3 = 3;

As you can see, it seems like a tedious task in order to just initialize and use the variables especially if they are used for the same purpose.

In Java and other programming languages, there is one capability wherein we can use one variable to store a list of data and manipulate them more efficiently. This type of variable is called an array.

An array stores multiple data items of the same data type, in a contiguous block of memory, divided into a number of slots.



☒ **Arrays two types** :

- One Diminution Array
- Multi Diminution Array

One Diminution Array

To declare an array, write the data type, followed by a set of square brackets[] , followed by the identifier name.

➔ Declaring Arrays

DataType []ArrayName;
or
DataType ArrayName[];

To **instantiate** (or create) **an array**, write the new keyword,followed by the square brackets containing the number of elements you want the array to have.

ArrayName = new DataType [SizeOfArray];

You can also **instantiate an array** by directly initializing it with data.

DataType ArrayName[] = {value1 , value2,… , valueN }

Example :
int arr[] = {1, 2, 3, 4, 5};

➔ This statement declares and instantiates an array of integers with five elements (initialized to the values 1, 2, 3, 4, and 5).

☒ Accessing an Array Element :

To access an array element, or a part of the array, you use a number called an **index** .

index number :

- ➔ assigned to each member of the array, to allow the program to access an individual member of the array.
- ➔ begins with zero and progress sequentially by whole numbers to the end of the array.

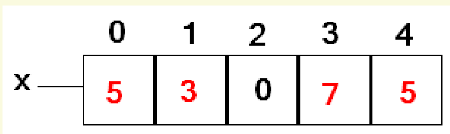
NOTE 1: Elements inside your array are from 0 to (sizeOfArray-1).

NOTE 2: once an array is declared and constructed, the stored value of each member of the array will be initialized to zero for number .

To accessing code , you can call from index , such as arr[0] = 5 ;

Example :

```
public class Arr
{
        public static void main(String [] arg)
        {
        int x [] new int [5];
        x[0] = 5 ;
        x[1]= 3 ;
        x[3] = 7 ;
        x[4]= x[0];
}}
```



Remember : if you declare an array with integer , all of array that initial values by 0's , double 0.0 …

Example:
Write program in Java enters 6 integer numbers, and print it …
Hint: Use an Arrays to complete it

```
import java.util.Scanner ;

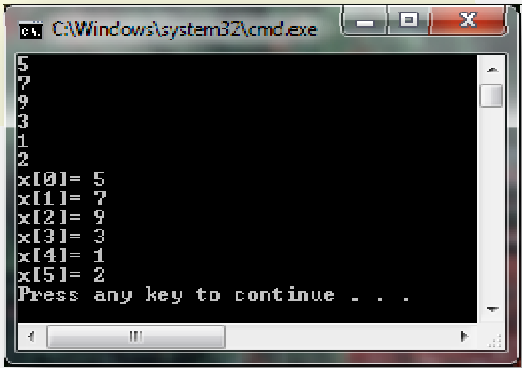
public class Arr
{
		public static void main(String [] arg)
		{
		Scanner input = new Scanner (System.in);
		int x [] = new int [6];

		for (int I = 0 ; I < 6 ; I++)
		x[I] = input.nextInt();

		for (int I = 0 ; I < 6 ; I++)
		System.out.println("x[" + I + "]= " + x[I]);

		}

}
```



- **Multidimensional Arrays** :

→Multidimensional arrays are implemented as arrays of arrays.
→Multidimensional arrays are declared by appending the appropriate number of bracket pairs after the array name.

➔ **Declaring Arrays** :

```
DataType ArrayName [][] = new DataType [row][col] ;
```

To **access an element** in a multidimensional array is just the same as accessing the elements in a one dimensional array.

Example :

```
public class Arr
{
            public static void main(String [] arg)
            {
            int x [][] = new int [3][2];
            x[0][0] = 5 ;
            x[0][1] = 3 ;
            x[1][1] = 7 ;
            x[2][0] = 9 ;
            x[2][1] = x[0][0] ;

            }

}
```

X	0	1
0	5	3
1	0	7
2	9	5

☒ Array Sorting :

you can sort an array ascending or descending as you want …

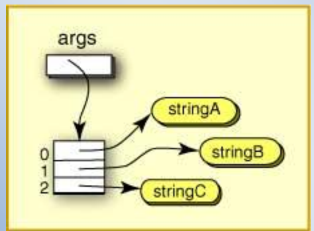
➔ in one Diminution array you can use for loop and variable to sort this , or

```
Arrays.sort(ArrayName);
```

➔ in two diminution there many methods to complete it , and you can convert it to one diminution array and sort one diminution , after this you can **convert to two diminution array** !! .

```
public class ClassName
{
        public static void main (String [] arg)
        {
        Statements ;
        }
}
```

The phrase String[] args says that main() has a parameter which is an array of String references. This array is constructed by the Java system just before main() gets control. The elements of the array refer to Strings that contain text from the command line .that starts the program



Example :

```
import java.util.*;
 public class java
 {
 public static void main(String [] arg)
 {

int a1[][] = new int[3][3];
int a2[] = new int[9];
 Scanner s=new Scanner(System.in);


System.out.println("PLZ. Enter values of array [3][3] :");
//insert values to two dimintion array
for (int i = 0 ; i < 3 ; i++)
    for (int j = 0 ; j< 3 ; j++)
        a1[i][j] = s.nextInt();



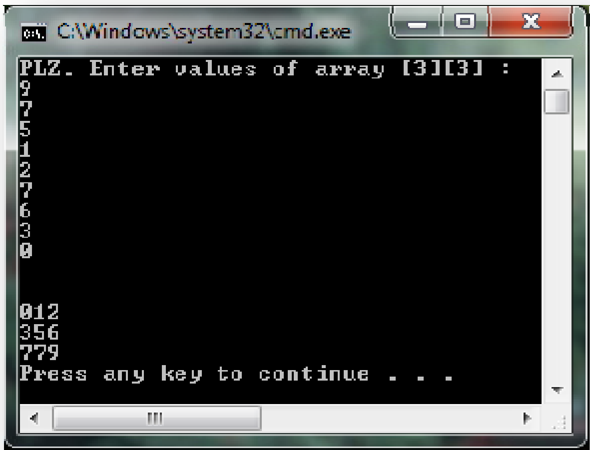
//convert two dimintion array to one dimintion
int index = 0;
for (int i = 0 ; i< 3 ; i++)
{
    for (int j = 0 ; j < 3 ; j++)
    {
    a2[index++] = a1[i][j];
    }
}

//sorting array
Arrays.sort(a2);


//convert one dimintion array to 2-Dimintion
index = 0 ;
for (int i = 0 ; i< 3 ; i++)
{
    for (int j = 0 ; j < 3 ; j++)
    {
      a1[i][j]=a2[index++];
    }
}



//printing array with two dimintion
System.out.println();
System.out.println();
 for(int i=0; i< 3 ;i++)
 {
     for(int j=0; j< 3;j++)
    System.out.print(a1[i][j]);
    System.out.println();
}
```



Object Oriented Programming

Inheritance

Ehab A. Qad'oumi ...

Reusability- building new components by utilizing existing components .. It is always good / “productive” if we are able to reuse something that is already exists rather than creating the same all over again. This is achieve by creating new classes, reusing the properties of existing classes.

This mechanism of deriving a new class from existing / old class is called "**inheritance**”.. The old class is known as “**base**” class, “**super**” class or "parent” class”; and the new class is known as “**sub**” class, “**derived**” class, or “child” class.

The inheritance allows subclasses to inherit all properties (variables and methods) of their parent classes.

➔ A subclass/child class is defined as follows:

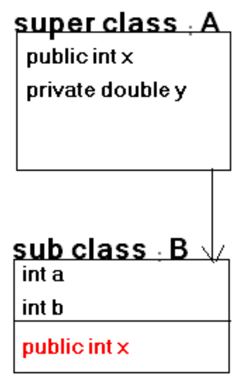
class SubClassName ***extends*** *SuperClassName*
{
fields declaration;
methods declaration;
}

The keyword “**extends**” signifies that the properties of super class are extended to the subclass. That means, subclass contains its own members as well of those of the super class. This kind of situation occurs when we want to enhance properties of existing class without actually modifying it.

Example :

```
class A
{
        public int x ;
        private double y ;
}

class B extends A
 {
        int a ;
        int b ;
 }
```



Example :
This example shows how ti make Inheretance with more than class .

```
 class Member
{
          public  int id;
          private String name;

          public Member () //Default constructor
          {
                    id = -1;
                    name="-";
          }
}

class Employee extends Member
{
          double salary;
}

class student extends Member
{
      double  avg;
}

public class java_inh
{
          public static void main(String [] arg)
          {
          Employee obj1 = new Employee ();
          obj1. id = 10 ;
          System.out.print(obj1.id ); // 10
          student obj2 = new student ();
          obj2.avg = 10 ; // avg has private access in student
          }
}
```

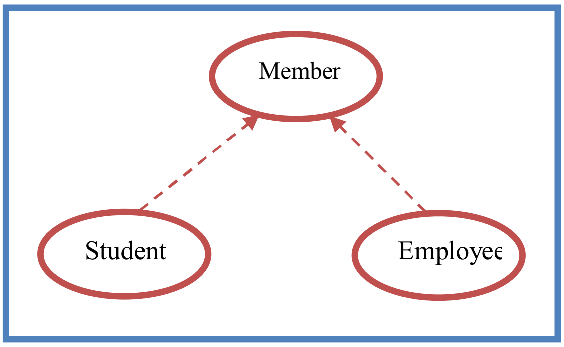
Access Modifier :

- ✓ **public** and **no modifier** : the same way as used in class level.
- ✓ **private** : members can only access.
- ✓ **protected** : can be accessed from 'same package' and a subclass existing in any package can access.

Constructor can't be inherit

only **one public class** in code , and the program Save as in that name .

Mostly , Object is created from the subclass . Where it contains data as well as data of Super Class .



Example :

```
 class A
{
            public int x ;

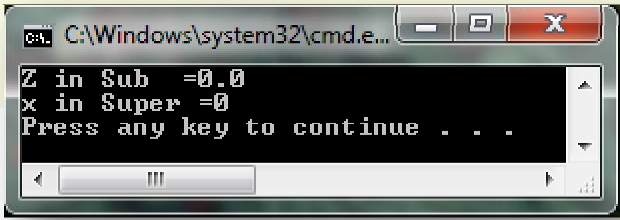
            public void print()
            {
            System.out.println("x in Super =" + x);
            }
}

class B extends A
{
            public double z ;
            public void print()
            {
            System.out.println("Z in Sub  =" + z);
            super. print() ;
            }
}

public class Test
{
            public static void main (String [] arg)
            {
            B obj = new B() ;
            obj.print();
            }
}
```

There are two method that named "print" in class A and B …

If you want to call something what is in the Super Class such as class A , calling him by " **super** " word Keyword .



Strings

Strings ,,, which are widely used in Java programming , are a sequence of characters. In the Java programming language, strings are objects.

The Java platform provides the String class to create and manipulate strings.

→ String : is a reference data type .
→you will insert text inside Double Quotes

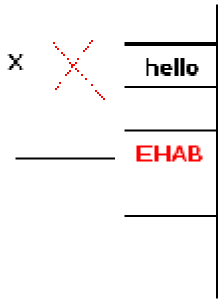
➔ Declare Strings :

```
String Var.Name = "Text" ;

String Var.Name = new Var.Name ("Text");
```

Note:

The String class is **immutable**, so that once it is created a String object cannot be changed. The String class has a number of methods, some of which will be discussed below, that appear to modify strings. Since strings are immutable, what these methods really do is create and return a new string that contains the result of the operation.




```
String x = "hello" ;
        x = "EHAB" ;
```

As with any other object, you can create String objects by using the new keyword and a constructor. The String class has thirteen constructors that allow you to provide the initial value of the string using different sources, such as an array of characters.

Example :

```
char arr[] = {'w','e','l','c','o','m','e'};
String ad_ar = new String (arr);
System.out.print(ad_ar);    //welcome
```

☒ String Methods :

➔ there are a lot of methods of String, I will display the most

Method Summary

char	charAt(int index) Returns the character at the specified index.
int	compareTo(Object o) Compares this String to another Object.
String	concat(String str) Concatenates the specified string to the end of this string.
boolean	equals(Object anObject) Compares this string to the specified object.
boolean	equalsIgnoreCase(String anotherString) Compares this String to another String, ignoring case considerations.
boolean	endsWith(String suffix) Tests if this string ends with the specified suffix.
boolean	startsWith(String prefix) Tests if this string starts with the specified prefix.
int	length() Returns the length of this string.
String	toLowerCase() Converts all of the characters in this String to lower case using the rules of the default locale.
String	toUpperCase() Converts all of the characters in this String to upper case using the rules of the default locale.

Example :

```
public class String_methods
{
        public static void main (String [] arg)
        {
        String s1 = "University" ;
        String s2 = "uNiVeRsItY";
        String s3 = "Unooo" ;
        String s4 = " Of Jordan";

         System.out.println("University");
        System.out.println("0123456789");
        System.out.println();

        System.out.println("the char at index of 3 is " + s1.charAt(3));
        System.out.println(s1.compareTo("University"));
```

Returns the character at the specified index. An index ranges from 0 to length() - 1. The first character of the sequence is at index 0, the next at inde .1, and so on, as for array indexing

The comparison between the string and a string of other depending on the **ASCII code** :
a→ 97 ; A → 65 ; SPACE → 32 ; 0 → 48

Compares this String to another String ,Two strings are considered equal case if they are of the same length, and corresponding characters in the two strings

```java
        System.out.println("s1 + s4 = " + s1.concat(s2) );

        System.out.println("s1 ? = s2 :" + s1.equals(s2));

        System.out.println("s1 ? = s2 :" + s1.equalsIgnoreCase(s2));

        System.out.println("s3 End with oo :" + s3.endsWith("oo"));

        System.out.println("s4  Start with of :" + s4.startsWith("of"));

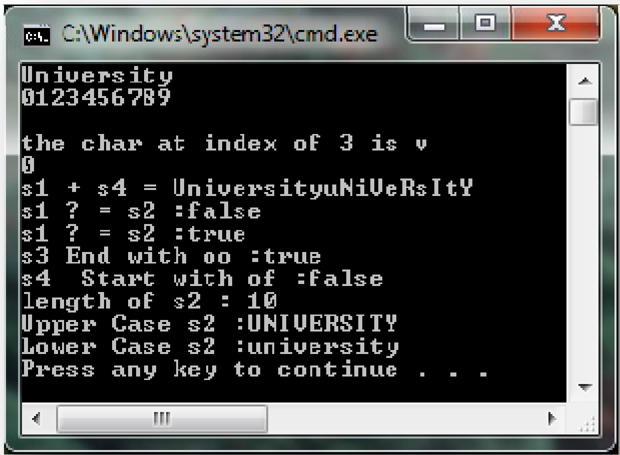
        System.out.println("length of s2 : " + s2.length());

        System.out.println("Upper Case s2 :" + s2.toUpperCase());

        System.out.println("Lower Case s2 :" + s2.toLowerCase());

    } //end of main
} // end of public class
```

endWith : Check if end of String equals the letters or not ,, that return true if yes or false if no .
startWith: : Check if initial of String equals the letters or not ,, that return true if yes or false if no



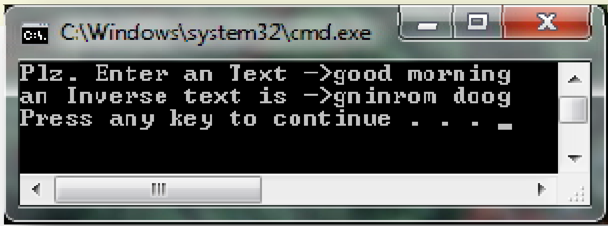
Example :

write a program in java that insert text from user and print reveres !

```
import java.util.* ;

 public class text_reveres
{
            public static void main (String [] arg )
         {
                        Scanner input = new Scanner (System.in);
                        System.out.print("Plz. Enter an Text ->");
                        String text = input.nextLine ();

                        System.out.print("an Inverse text is ->") ;
                        for (int i = text.length()-1 ; i >=0 ; i--)
                        System.out.print (text.charAt(i));
                        System.out.println();
           }
}
```

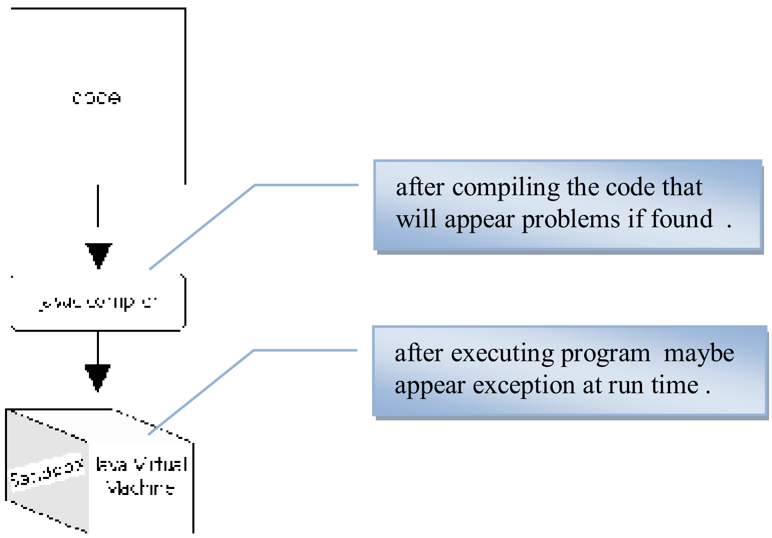


Exception Handling

Programmers in any language endeavor to write bug-free programs, programs that never crash, programs that can handle any circumstance with grace and recover from unusual situations without causing a user any undue stress. Good intentions aside, programs like this don't exist.

In real programs, errors occur because programmers didn't anticipate possible problems, didn't test enough, or encountered situations out of their control-bad data from users .

Errors usually occur because of a lack of efficiency of the code or the same user Bad data entry …



- **Runtime exceptions:** A runtime exception is an exception that occurs that probably could have been avoided by the programmer. As opposed to checked exceptions, runtime exceptions are ignored at the time of compilation.

- **Errors:** These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Errors are typically ignored in your code because you can rarely do anything about an error. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

☒ Types of Exceptions :

- Out of bounds
- Input mismatch exception
- Arithmetic exception
- Null pointer exception
- ………

- **OutOfBoundsException** : This exception occurs when you out the Array for the specified limit …

Example :

```
int x[] = {4,5,6};
x[1] = 15 ;
x[3] = 10 ;  //exception
```

- **InputMismachException** : This exception occurs when you enter the wrong value to a variable through input from user …

Example :

```
Scanner input = new Scanner (System.in);
int y = input.nextInt(); //→5.3  exception
```

- **ArithmaticException** : This exception occurs when you divided by zero …

Example :

```
Scanner input = new Scanner (System.in);
int x = input.nextInt();
int y = input.nextInt();
int z = x/y ; // if you enter y =0 , exception will be appear .
```

- **NullPointerException** : This exception occurs when you not create an object from class …

Example :

```
Scanner input ;
int x = input.nextInt(); // exception
```

Note :
Previous problems can be solved with numerous ways including one that he put a warning message to the user or using control sentences if () …

☒ Catching Exceptions

A method catches an exception using a combination of the try and catch keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch looks like the following:

```
try
{
        Statements that may generate exception
}
catch (ExceptionType e)
{
        Statement to process exception
}
.
.
.
.
finally
{
        Statement to release resources .
}
```

- Encloses the code that may generate an exception
- if the exception occurs , the remaining code in the try block will be skipped .
- a try block must be following by at least one catch or finally block .

- each try block is followed by one or more catch blocks
- each catch block specifies the type of exception if can catch .
- each catch block have a single parameter only

- finally block of code **always executes**, whether or not an exception has occurred.
- it is optional appear after the last catch block if one exist.

Example :

```
import java.util.Scanner ;
public class Test
{
        public static void main (String [] arg)
        {
           Scanner input = new Scanner (System.in);

           System.out.print("Enter the first number : ");
           int x = input.nextInt();
           System.out.print("Enter the Second number : ");
           int y = input.nextInt();

           int z = x/y ;

           System.out.print("The resoult is :" + z);
        }
}
```

- if the user input x = any integer number and y = any integer number except 0 ,,, it's Ok ..
- if the user input any number not integer or input y = 0 !! that is exception occurs

Example :

```
import java.util.Scanner ;
import java.util.InputMismatchException;
public class Test
{
        public static void main (String [] arg)
        {
          try
          {
                    Scanner input = new Scanner (System.in);
                    System.out.print("Enter the first number : ");
                    int x = input.nextInt();
                    System.out.print("Enter the Second number : ");
                    int y = input.nextInt();
                    int z = x/y ;
                    System.out.println("The resoult is :" + z);
          }

          catch (ArithmeticException e)
          {
                    System.out.println("you can't divided by zero !");
          }

          catch (InputMismatchException e)
          {
                    System.out.println("Can't be casting !");
          }

          finally
          {
                    System.out.println("finally always execute !");
          }

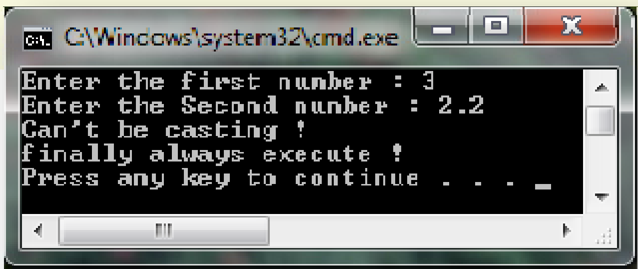
        }
}
```

If you want to use " InputMismatchException " you will import the class InputMismatchException from **java.util package** .

if the user input an integer number in x and will enter an integer number in y except 0 . that's Ok

if the user input an integer number in x and enter 0 in y ,,, exception occurs , will go to " ArithmeticException " catch block and go to execute finally and go out of try statement .

if the user enter not an integer number in x or y , exception occurs and go to " InputMismatchException " catch block and go to execute finally and go out of try statement .



Ehab A. Qad'oumi ...

☒ Exception Word :

If none of the previous catch block are passing , you put word "**Exception** " , they implemented in the event there was no Block is winner …

→Must put it block after all catch block's and before finally if found …

```
import java.util.Scanner ;
import java.util.InputMismatchException;
public class Test
{
        public static void main (String [] arg)
        {
          try
          {
                    Scanner input = new Scanner (System.in);
                    System.out.print("Enter the first number : ");
                    int x = input.nextInt();
                    System.out.print("Enter the Second number : ");
                    int y = input.nextInt();
                    int z = x/y ;
                    System.out.println("The resoult is :" + z);
          }

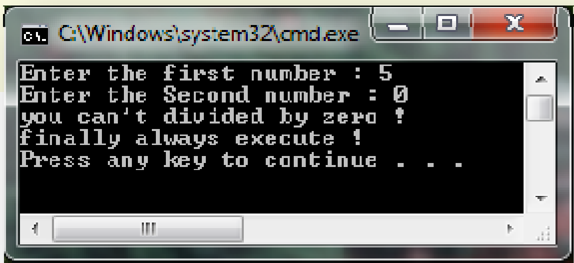
          catch (InputMismatchException e)
          {
                    System.out.println("Can't be casting !");
          }

          catch (Exception e)
          {
                    System.out.println(" Exception block execute") ;
          }

          finally
          {
                    System.out.println("always execute !");
          }

        }
}
```

Note :
it must be the last catch block and before finally .



Java Applet

☒ Types of Java Program (SE) :

1. Application Programs .
2. **Applet Programs** .

→ **Application programs** : are stand-alone programs that are written to carry out certain tasks on local computer such as solving equations, reading and writing files etc. (must Contain main method)
The application programs can be executed using two steps

1. Compile source code to generate Byte code using Javac compiler.
2. Execute the byte code program using Java interpreter.

→ **Applet programs:** Applets are small Java programs developed for Internet applications. An applet located in distant computer can be downloaded via Internet and executed on a local computer using Java capable browser .(maybe contain main method) .

☒ **Java development process :**

The javac command compiles Java source code (.java) into *bytecode* (.class).



These bytecodes are loaded and executed in the Java virtual machine (JVM), which is embeddable within other environments, such as Web browsers and operating systems.

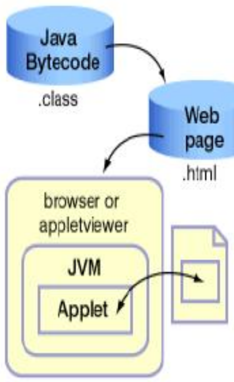


☒ Displaying applets :

An apple is a Java program that is referenced by a Web page and runs inside a Java-enabled Web browser.

An applet begins execution when an HTML page that "contains" it is loaded.

Either a Java-enabled Web browser or the applet viewer is required to run an applet.



Note : We Will Use **JFrame** class in Programming .

the screen of windows

➔ The Syntax to write code applet :

```
import java.awt.Graphics;
import java.awt.Color ;
import  javax.swing.JFrame ;
public class ClassName extends JFrame
{
        public ClassName ()
        {
                setTitle("String");
                SetSize(Width , Higth);
                SetVisible(Boolean var);
        } //end constructor

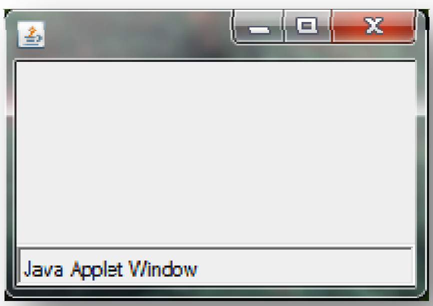
        public void paint(Grapgics g)
        {
                super.paint(g);
                .
                g.methods () ;
                .
        } //end paint method
} //end of public class
```

The **awt** provides **user interface components** , such as Label , Button , List ,... But it does not contain ingredients for Graphics such as Line , Rectangle , Oval...

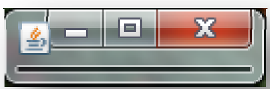
Therefore we must inherit this the class using something called Overriding …. Where we can use the library is ready and the amendment to the components .

the constructor : will must contain three methods :

1. **setTitle("String")** : Determine the Title shown on the title bar ,if you don't determine this , the default shown no title .



2. **setVisible(boolean)** : Control the show and did not show the form to view the results , if you don't determine it , the default is false .
3. **setSize(Wedth , Height)** : Control the size of the form , if you don't determine the size , the size is (0,0) .



☒ Applet methods :

- Draw Line :
to draw a **line** : `g.drawLine ( x, y, z, w );`

→ Draws a line, between the points (x1, y1) and (x2, y2) in this graphics context's coordinate system.

-Draw **String** :
to draw a string : `g.drawString("String", x ,y);`

→Draws the text given by the specified string, x and y is the start position of draw string **.**

- Draw a **rectangle**
to draw the **outline** of the specified of Rectangle :

```
g.drawRect( x, y, width, height );
```

to draw the **fill color** Rectangle : `g.fillRect( x, y, width, height );`

-Draw an **Oval** :
to draw the **outline** of the specified of Oval :

```
g.drawOval( x, y, width, height );
```

to draw the **fill color** Oval : `g.fillOval( x, y, width, height );`

→in **Rectangle** and **oval** , x and y is the initial values , width and height is the diminutions of rectangle .

-Draw **Rounded Rectangle** :
to draw the **outline** RoundRect :

```
g. drawRoundRect(x,y, width, height , arcX , arcY);
```

to draw the **fill color** RoundRect :

```
g. fillRoundRect(x,y, width, height , arcX , arcY);
```

→The drawRoundRect method draws rectangles with rounded edges and it requires six arguments . The first four work the same as for normal rectangles. The last pair of numbers determine how far along the edges of the rectangle the arc for the corner will start. If the numbers are large, the rectangle has a more rounded appearance .

Note : fillRoundRect is the same ,but with fill color .

-Draw **Arc** :

to draw the **Arc** : `g. drawArc(x,y, w, z , startAngle , arcAngle );`

to draw the **fill color** Arc : `g. fillArc(x,y, w, z , startAngle , arcAngle );`

→Draw the outline of a circular or elliptical arc covering the specified rectangle. and it requires six arguments . The first four work the same as for normal Oval. The last pair of numbers determine the beginning angle. and the last arguments determine .

-Draw **PolyLine** :

to draw the **boarder** of polyLine : `g.drawPolyline (arrayX[] , array[] , nPoints );`

→Draws a sequence of connected lines and not closed shape defined by arrays of *x* and *y* coordinates. Each pair of (*x*, *y*) coordinates defines a point. The figure is not closed if the first point differs from the last point. there are three parameter the first and second one is an array of *x*,y points and the last one is the total number of points .

-Draw **Polygon** :

to draw the **outline** of polygon : `g.drawPolygon(arrayX[] , array[] , nPoints );`

to draw the **fill color** polygon : g.fillPolygon(arrayX[] , array[] , nPoints);

→Draws a sequence of connected lines and closed shape defined by arrays of *x* and *y* coordinates. Each pair of (*x*, *y*) coordinates defines a point. there are three parameter the first and second one is an array of *x*,y points and the last one is the total number of points .

Note that : you can fill color the polygon because it a close shape but you can't fill color a polyline .

- clear Rectangle :

to clear an area of rectangle : `g.clearRect(x , y , width , height );`

→Clears the specified rectangle by filling it with the background color of the current drawing surface.

☒ Color Class :

to use Colors in Java Applet , you will use Color Class and import it by :

```
import java.awt.Color
```

- ➔ there are RGB Type of Colors , every color contain 256 of Degree (0-255) .
- ➔ many methods to use a Color Class :
  - ✓ setColor(Color.ColorName);
    - → Color.red
    - → Color.blue
    - → Color.black
    - → Color.ColorName
  - ✓ Color ObjectName = new Color (D_O_R , D_O_G , D_O_B);

    setColor(ObjectName) ;

After you select a color, the color will not be applied unless use : setColor (obj);

Note :
If you not use a Color Class to change the color : the Default color is black .

Example :

```
import java.awt.Color ;
import java.awt.Graphics ;
import javax.swing.JFrame;
public class Drawing extends JFrame
{

        public Drawing()
        {
        setTitle("Applet");
        setSize(300,300);
        setVisible(true);
        } //end constructor

          public void paint(Graphics g )
          {
          super.paint(g) ;

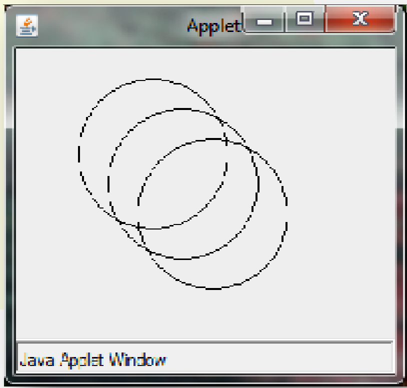
          g.drawOval(50,50,100,100);
          g.drawOval(70,70,100,100);
          g.drawOval(90,90,100,100);

          } //end paint

} //end drawing calss
```

if you need to use the Color library , you will import Color class :
import java.awt.Color ;

I prefer to be in this **order** so that there will be no problems in the presentation of results .



Simple graphic + Simple graphic = Big and Nice graphic ☺

Example :

```
import java.awt.Color ;
import java.awt.Graphics ;
import javax.swing.JFrame;
public class Drawing extends JFrame
{

        public Drawing()
        {
        setTitle("Applet");
        setSize(220,430);
        setVisible(true);
        } //end constructor

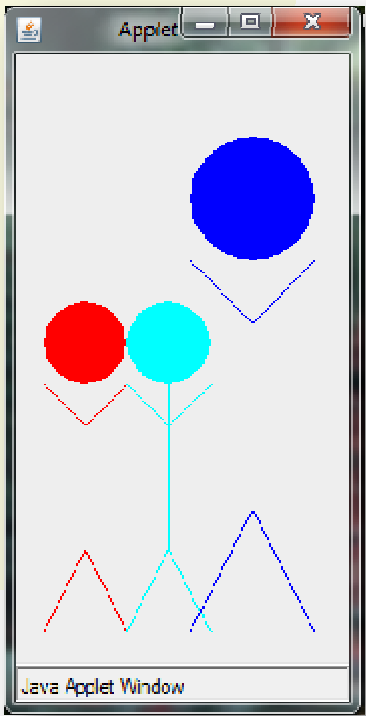
          public void paint(Graphics g )
          {
          super.paint(g) ;

          g.setColor(Color.red);
          g.fillOval(25, 180, 50, 50);
          g.drawLine(50, 230, 50, 330);
          g.drawLine(50, 330, 25, 380);
          g.drawLine(50, 330, 75, 380);
          g.drawLine(50, 255, 25, 230);
          g.drawLine(50, 255, 75, 230);

          g.setColor(Color.cyan);
          g.fillOval(75, 180, 50, 50);
          g.drawLine(100, 230, 100, 330);
          g.drawLine(100, 330, 75, 380);
          g.drawLine(100, 330, 125, 380);
          g.drawLine(100, 255, 75, 230);
          g.drawLine(100, 255, 125, 230);

          g.setColor(Color.blue);
          g.fillOval(113, 80, 75, 75);
          g.drawLine(150, 155, 150, 305);
          g.drawLine(150, 305, 113, 380);
          g.drawLine(150, 305, 187, 380);
          g.drawLine(150, 193, 113, 155);
          g.drawLine(150, 193, 187, 155);
          } //end paint

} //end drawing calss
```



Example : Draw Monaleza

```
import java.awt.Color ;
import java.awt.Graphics ;
import javax.swing.JFrame;
public class Monalisa extends JFrame
{

        public Monalisa()
        {
        setTitle ("Monaleza ! ");
        setSize(400,400);
        setVisible(true);
        }

        public void paint(Graphics g)
        {
        super.paint(g);
        Color c1 = new Color(255,210,145);
        g.setColor(c1);
        g.fillRoundRect(147,84,103,74,23,23);
        g.fillOval(147,94,103,132);
        g.setColor(Color.black);

int[] hairX = { 125,135,150,220,270,315,245,235,200,160,150,155,180,190,125};
int[] hairY = { 315,120,75,60,95,290,320,120,90,90,135,200,230,260,315};

        g.fillPolygon(hairX,hairY,15);

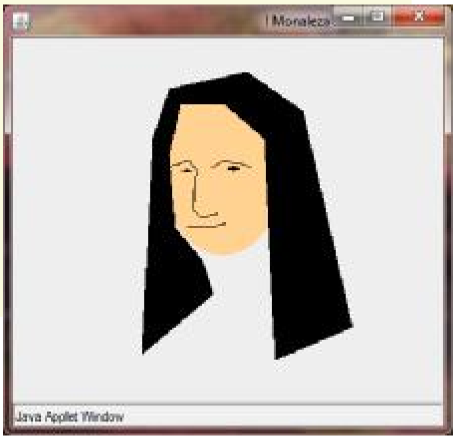
        int[] eyebrow1X = { 151, 168, 174, 171, 178, 193 };
        int[] eyebrow1Y = { 145, 140, 148, 184, 191, 188 };
        g.drawPolyline(eyebrow1X, eyebrow1Y, 6);

        int[] eyebrow2X = { 188, 197, 213, 223 };
        int[] eyebrow2Y = { 146, 141, 142, 146 };
        g.drawPolyline(eyebrow2X, eyebrow2Y, 4);

        int[] mouthX = { 166, 185, 200 };
        int[] mouthY = { 199, 200, 197 };
        g.drawPolyline(mouthX, mouthY, 3);

        g.fillOval(161,148,10,3);
        g.fillOval(202,145,12,5);

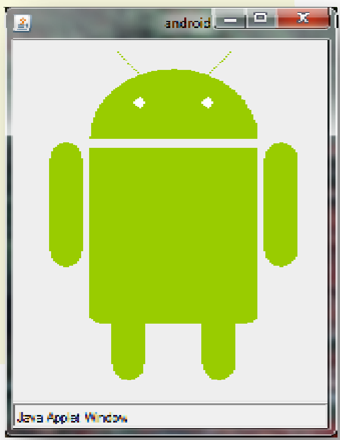
} //end Monalisa calss
```



Example : Draw Android Logo

```
import java.awt.Color ;
import java.awt.Graphics ;
import javax.swing.JFrame;
public class Drawing extends JFrame
{
        public Drawing()
        {
        setTitle("android");
        setSize(300,400);
        setVisible(true);
        } //end constructor

        public void paint(Graphics g )
        {
        super.paint(g) ;
        g.setColor(new Color(153, 204, 0));
        g.fillArc(75, 55, 150, 125, 0, 180);
        g.fillRect(75, 125, 150, 150);
        g.fillRoundRect(75, 260, 150, 20, 20, 20);
        g.fillRoundRect(40, 120, 30, 110, 30, 30);
        g.fillRoundRect(230, 120, 30, 110, 30, 30);
        g.fillRoundRect(95, 260, 30, 70, 30, 30);
        g.fillRoundRect(175, 260, 30, 70, 30, 30);
        g.drawLine(125, 65, 100, 40);
        g.drawLine(175, 65, 200, 40);
        g.setColor(Color.white);
        g.fillOval(115, 80, 10, 10);
        g.fillOval(175, 80, 10, 10);
        } //end paint
} //end drawing calss
```



~(خِتَامُهُ مِسْكٌ وَفِي ذَلِكَ فَلْيَتَنَافَسِ الْمُتَنَافِسُونَ)~

وهكذا ترنم القلم على قيثارة الفكر والشجن ، متجولا حينا ، ومتأملا أحيانا ؛ فالموضوع كالدوحة المثمرة ، أغصانها وارفة ، وثمارها متعة لذيذة ، فحقا تحتاج إلى صفحات وصفحات كي نأتي على ثمارها ، فما بالنا بظلالها الوارفة ...

هذا جهد متواضع ، لعله أنار غصنا من أغصانها ، وهفا عبر أشجان وأفكار ...

لا يسعني إلا أن أقول الحمد والشكر لله ... الذي أعانني على هذا العمل ...
ولا أقول كتابي كاملا ... فلا يوجد شي كامل سوى الله ...

~(وَمَا أُوتِيتُمْ مِنَ الْعِلْمِ إِلَّا قَلِيلًا)~

لا تنسوني من صالح دعائكم ...

ايهاب القدومي ...

Alqadoumi_ehab@yahoo.com

https://www.facebook.com/sadlover91

https://twitter.com/ehab_qadoumi

+962788124977

Ehab A. Qad'oumi ...